Documento de propuesta de desarrollo de software I, II y III

DUNGEON SCHOOL

Emilio José Plaza Valdés
Maria Lucia Dilso Mosquera
Brayan Andres Ramos Fuentes
Diego Andrés Salas Álvarez

Tutor: Alexander Toscano Ricardo

Descripción del videojuego educativo

Se busca crear y diseñar un videojuego original para ordenadores ya sea flash o instalable que permita la evaluación y diseño de estas en diferentes áreas centradas en temas específicos.

Dichas evaluaciones dependiendo el área estarán divididas en 2 sesiones donde en la primera se centrarán en preguntas esparcidas por un mapa que parodia a un salón de clases donde también abran ítems de ayuda de manera aleatoria y la segunda sección se centrará en la representación de un maestro como jefe final donde estará el grueso de la evaluación que se está aplicando de igual manera se podrán usar ítems sobrantes de primera sección en la parte del jefe.

El videojuego tendrá temas y preguntas prefabricadas y de diferentes dificultades de igual manera se buscará en próximas actualizaciones que estas puedan ser más personalizables al igual que los temas centrales o que se implemente la capacidad de integrada por IA de generar preguntas pertinentes en base a un plan de periodo o plan de clases.

Etapa 1 Diseño de la Aplicación y Análisis de Requisitos

1. Introducción

Propósito del Documento

El presente documento tiene como finalidad documentar el proceso de diseño, análisis e implementación del videojuego educativo en su etapa inicial. El desarrollo se basará en la metodología SECMALI, la cual permitirá estructurar y validar cada fase del proyecto para garantizar su efectividad pedagógica y técnica.

El documento se divide en tres etapas para facilitar su entendimiento y aplicación en la asignatura de diseño de software.

Etapa 1 Diseño de la Aplicación y Análisis de Requisitos

Diseño de la Aplicación y Análisis de Requisitos, Esta fase se enfoca en definir los elementos fundamentales del videojuego educativo, considerando su estructura, mecánicas y requerimientos técnicos. El objetivo es establecer una base sólida que permita desarrollar una experiencia lúdica centrada en el aprendizaje y la evaluación de conocimientos en un entorno simulado tipo aula escolar.

El videojuego estará dividido en dos secciones principales:

1. **Exploración del mapa:**

El jugador recorrerá un entorno en pixel art que representa un aula de clases. A lo largo del mapa encontrará preguntas educativas de dificultad variada, así como ítems de ayuda dispersos aleatoriamente que podrán ser usados más adelante.

2. **Enfrentamiento con el maestro (jefe final):**

Tras completar la exploración, el jugador se enfrentará al “jefe final”, representado por un maestro. En esta fase se evaluarán los conocimientos adquiridos mediante preguntas más complejas, y se podrán utilizar los ítems recolectados previamente para facilitar el progreso.

Objetivos del diseño:

- Evaluar el aprendizaje de los estudiantes de forma dinámica y entretenida.
- Fomentar el uso de la tecnología como recurso pedagógico.
- Motivar a los usuarios mediante retos progresivos y retroalimentación visual.
- Crear un entorno inmersivo adaptado a la estética pixel art.

Requerimientos funcionales:

- Sistema de preguntas por niveles o temas.
- Registro de respuestas correctas e incorrectas.
- Uso estratégico de ítems recolectados.
- Interfaz intuitiva para navegación y combate.
- Escenarios interactivos con diseño en pixel art.

Requerimientos no funcionales:

- Compatibilidad con ordenadores de bajos recursos.
- Bajo consumo de ancho de banda en caso de conexión al servidor.
- Facilidad de actualización y personalización de preguntas.
- Diseño atractivo y accesible para estudiantes de distintos niveles.

Etapa 2: Persistencia de Datos con Backend – Servidor

En esta etapa se desarrollará el backend que permitirá almacenar y gestionar la información generada durante el uso del videojuego. Esto incluye el progreso de los jugadores, respuestas seleccionadas, estadísticas de juego y configuraciones personalizadas.

Objetivos:

- Crear una API que permita el almacenamiento y recuperación de datos del juego.
- Implementar una base de datos para guardar el progreso del jugador, resultados por nivel, preguntas y configuración de ítems.
- Facilitar la personalización futura del contenido por parte de docentes (subida de preguntas propias, configuraciones de dificultad).

Componentes:

- Endpoints para el registro y login de usuarios.
- Registro de progreso por sesión.
- Almacenamiento de preguntas y resultados por jugador.
- Carga dinámica de datos (preguntas, niveles, ítems).

Tecnologías sugeridas:

- Lenguaje: JavaScript o TypeScript.
- Framework: Express.js o Nest.js.
- Base de datos: MongoDB (NoSQL) o PostgreSQL (SQL), según necesidades.
- API: RESTful con autenticación mediante JWT.
- Herramientas adicionales: Mongoose (si se usa MongoDB), Prisma (si se usa PostgreSQL).

Etapa 3: Consumo de Datos y Desarrollo Frontend – Cliente

Esta etapa contempla la implementación de la interfaz visual y funcional del videojuego en estilo pixel art, centrado en brindar una experiencia lúdica y educativa atractiva.

Objetivos:

- Desarrollar una interfaz gráfica en pixel art que represente un aula y escenarios de batalla con el jefe final.
- Implementar mecánicas de exploración, interacción con preguntas y uso de ítems.
- Conectar el cliente con el backend para cargar preguntas, enviar respuestas y registrar resultados.

Características:

- Sección de exploración estilo RPG 2D con estética pixel art.
- Mecánicas para recolectar ítems y resolver preguntas en un entorno interactivo.
- Sección de jefe final donde se aplican preguntas más complejas.
- Uso de los ítems recolectados como ayudas o potenciadores.
- Feedback visual retro acorde a la estética pixel art (barras de energía, cuadros de texto tipo RPG).

Tecnologías sugeridas:

- Motor/Framework para 2D en pixel art: Phaser.js o Godot (GDScript) si se quiere mantener libre de Unity.
- Lenguaje: JavaScript (Phaser) o GDScript (Godot).
- Recursos visuales: Sprites 2D en pixel art, tilesets para el mapa del aula, animaciones sencillas.
- Sonido: Efectos retro y música chiptune para ambientar.

Alcance del Proyecto Dungeon School

Este proyecto busca crear un videojuego educativo online que combine diversión con aprendizaje. La idea es que los estudiantes puedan recorrer un escenario tipo aula mientras responden preguntas y enfrentan retos, todo con un estilo visual en pixel art. El juego también incluye una parte final donde el jugador se enfrenta a un “maestro jefe” para demostrar lo que aprendió, como si fuera un juego de rol.

En este documento se explican las funciones principales que tendrá la primera versión del juego, además de algunas ideas que podrían agregarse en futuras actualizaciones para mejorarlo.

Funcionalidades actuales:

- Exploración de un mapa.
- Interacción con preguntas de diferentes niveles de dificultad.
- Recolección y uso de ítems como ayudas estratégicas.
- Enfrentamiento con un jefe final (maestro) mediante preguntas más complejas.
- Registro de respuestas correctas e incorrectas.
- Feedback visual retro (barras de energía, efectos de sonido chiptune, etc.)

Funcionalidades futuras:

- Personalización de pruebas por parte del docente.
- Generación automática de preguntas con IA según el plan de clase.
- Multijugador.
- Sistema de progresión con logros.
- Versión portable offline

Guardar el progreso del jugador automáticamente.

- Personalizar el avatar del estudiante con ropa o accesorios.
- Elegir entre distintos mapas escolares para explorar.

- Desbloquear niveles temáticos (ciencias, historia, etc.).
- Usar objetos especiales para resolver preguntas más rápido.
- Obtener pistas al ver contenido educativo dentro del juego.
- Subir puntuaciones a un ranking global o escolar.
- Participar en torneos académicos entre colegios.
- Chatear con otros jugadores dentro del aula virtual.
- Conseguir medallas por logros educativos (ej. responder 10 preguntas seguidas bien).
- Desbloquear zonas secretas si se alcanza cierto puntaje.
- Usar animaciones o reacciones del maestro jefe según el rendimiento.
- Escuchar narraciones o audios explicativos como ayuda extra.
- Cambiar la dificultad del juego según el nivel del estudiante.
- Integrar videos educativos antes de ciertas preguntas clave.
- Acceder a estadísticas personales de rendimiento (áreas fuertes y débiles).
- Permitir a los profes crear mini misiones con recompensas.
- Recibir retroalimentación automática según errores comunes.
- Jugar con teclado, mouse, pantalla táctil o incluso control.
- Compartir resultados o certificados con padres o docentes.
- Usar monedas virtuales para desbloquear contenido educativo extra.
- Añadir minijuegos entre preguntas para mantener la motivación.
- Personalizar el fondo o estilo del aula virtual.
- Descargar preguntas desde una nube educativa segura.
- Participar en eventos mensuales con recompensas especiales.
- Acceder a una biblioteca con guías de estudio integradas.

- Usar un sistema de energía para regular el tiempo de juego educativo.
- Hacer misiones diarias o semanales con objetivos específicos.
- Guardar partidas y continuar desde donde se dejó.
- Usar comandos por voz para responder o navegar (accesibilidad).

Definiciones y Acrónimos

- API: Interfaz de Programación de Aplicaciones (Application Programming Interface).
- DBMS: Sistema de Gestión de Bases de Datos (Database Management System).
- JWT: Token Web JSON (JSON Web Token), utilizado para autenticar usuarios.
- CRUD: Crear, Leer, Actualizar y Borrar (Create, Read, Update, Delete).
- Pixel Art: Estilo gráfico basado en píxeles visibles, común en videojuegos retro.
- RPG: Juego de Rol (Role-Playing Game), estilo de juego donde el jugador asume el papel de un personaje en una narrativa.

2. Descripción General

Objetivos del Sistema

El objetivo del sistema es proporcionar una pizarra compartida dentro de un Sistema de Gestión de Contenido llamado CREAVI que permita a los usuarios colaborar de manera eficiente y efectiva, facilitando la creación, visualización y edición de contenido visual en tiempo real. Esta pizarra compartida se diseñará con el propósito de mejorar la comunicación y la colaboración en un entorno en línea, ofreciendo a los usuarios una plataforma intuitiva y versátil para crear y compartir ideas, diagramas, esquemas y contenido visual de manera colaborativa, enriqueciendo así la experiencia de usuario y la productividad en el uso del CMS.

Funcionalidad General

- **Creación y Edición Colaborativa**: Permite a los usuarios crear y editar contenido en la pizarra de forma colaborativa en tiempo real. Múltiples usuarios pueden trabajar en el mismo documento simultáneamente.
- **Herramientas de Dibujo y Anotación**: Proporciona herramientas de dibujo, pinceles, formas y opciones de anotación que permiten a los usuarios plasmar sus ideas y conceptos de manera visual.
- **Carga de Imágenes y Multimedia**: Permite a los usuarios cargar imágenes, videos y otros medios directamente en la pizarra, lo que facilita la ilustración de conceptos.
- **Organización de Contenido**: Ofrece opciones para organizar y estructurar el contenido en la pizarra, como la creación de capas, agrupación de elementos y uso de etiquetas.
- **Historial de Revisiones**: Registra un historial de revisiones que permite a los usuarios rastrear los cambios realizados en la pizarra y restaurar versiones anteriores si es necesario.
- **Compartir y Colaborar**: Permite compartir la pizarra con otros usuarios a través de enlaces o invitaciones, lo que facilita la colaboración con colegas, clientes o amigos.
- **Comentarios y Chat en Tiempo Real**: Los usuarios pueden comentar y discutir sobre el contenido de la pizarra a través de un chat en tiempo real, lo que facilita la comunicación durante la colaboración.
- **Exportación e Impresión**: Ofrece la capacidad de exportar el contenido de la pizarra en varios formatos (PDF, imagen, etc.) y la opción de imprimirlo.
- **Integración con el CMS**: Se integra de manera transparente con el sistema de gestión de contenido (CMS CREAVI), lo que permite incrustar pizarras en los contenidos, metodologías o cualquier otro tipo de componente que permita la pizarra.
- **Personalización y Temas**: Permite a los usuarios personalizar la apariencia de la pizarra y seleccionar temas que se adapten a sus necesidades.
- **Acceso Seguro**: Proporciona medidas de seguridad para garantizar que solo los usuarios autorizados puedan acceder y editar la pizarra.
- **Notificaciones y Actualizaciones en Tiempo Real**: Los usuarios reciben notificaciones sobre cambios en la pizarra y pueden ver actualizaciones en tiempo real mientras otros editan.
- **Acceso Móvil**: Ofrece una experiencia de usuario optimizada en dispositivos móviles, permitiendo el acceso y la colaboración desde smartphones y tabletas.
- **Búsqueda y Filtros**: Facilita la búsqueda de contenido en la pizarra y la aplicación de filtros para organizar y encontrar información específica.

- **Gestión de Usuarios y Permisos**: Permite a los administradores gestionar usuarios y definir permisos de acceso y edición.
- **Informes y Analíticas**: Proporciona información sobre el uso de la pizarra, como quién la ha editado, cuándo se realizaron cambios y estadísticas sobre el contenido(XAPI).

Usuarios del Sistema

Los siguientes usuarios pueden interactuar con la pizarra dependiendo de las funcionalidades.

Funcionalidad	Administradores	Docente Investigador	Docente Invitado	Alumno	Invitado
Creación y Edición		✓	✓	✓	
Herramientas de Dibujo		✓	✓	✓	
Carga de Imágenes y Multimedia		✓	✓	✓	
Edicion de Imágenes y Multimedia		✓	✓	✓	
Organización de Contenido		✓	✓		
Historial de Revisiones	✓	✓	✓		
Compartir y Colaborar	✓	✓	✓		
Comentarios y Chat	✓	✓	✓	✓	✓
Exportación e Impresión		✓	✓	✓	
Integración con el CMS	✓	✓	✓		
Personalización y Temas	✓	✓	✓		
Acceso Seguro	✓	✓	✓	✓	
Notificaciones en Tiempo Real	✓	✓	✓	✓	
Acceso Móvil	✓	✓	✓	✓	✓
Búsqueda y Filtros	✓	✓	✓	✓	✓
Gestión de Usuarios y Permisos	✓				
Informes y Analíticas	✓	✓	✓		

Restricciones

Solo usuarios agregados por un anfitrión de la pizarra tendrán acceso a las funcionalidades descritas en la tabla anterior, un anfitrión puede agregar otros anfitriones a la pizarra quienes pueden ser docentes, alumnos o invitados, también se les puede dar el rol de moderador y/o administrador de pizarra. Las funcionalidades de estos dos roles no se han descrito aún.

3. Requisitos Funcionales

Creación y Edición de Contenido:

- Los usuarios deben poder crear y editar contenido en tiempo real en la pizarra.

- Los usuarios deben tener acceso a herramientas de dibujo, anotación y edición de contenido.

Compartir y Colaborar:

- Los usuarios deben poder compartir la pizarra con otros usuarios a través de enlaces o invitaciones.
- Múltiples usuarios deben poder colaborar en la misma pizarra simultáneamente.

Carga de Multimedia:

- Los usuarios deben poder cargar imágenes, videos y otros medios directamente en la pizarra.

Historial de Revisiones:

La aplicación debe mantener un historial de revisiones que permita a los usuarios rastrear cambios realizados en la pizarra y restaurar versiones anteriores.

Comentarios y Chat en Tiempo Real:

- Los usuarios deben poder comentar y discutir sobre el contenido de la pizarra a través de un chat en tiempo real.

Exportación e Impresión:

- Los usuarios deben poder exportar el contenido de la pizarra en varios formatos (PDF, imagen, etc.) y tener la opción de imprimirlo.

Integración con el CMS:

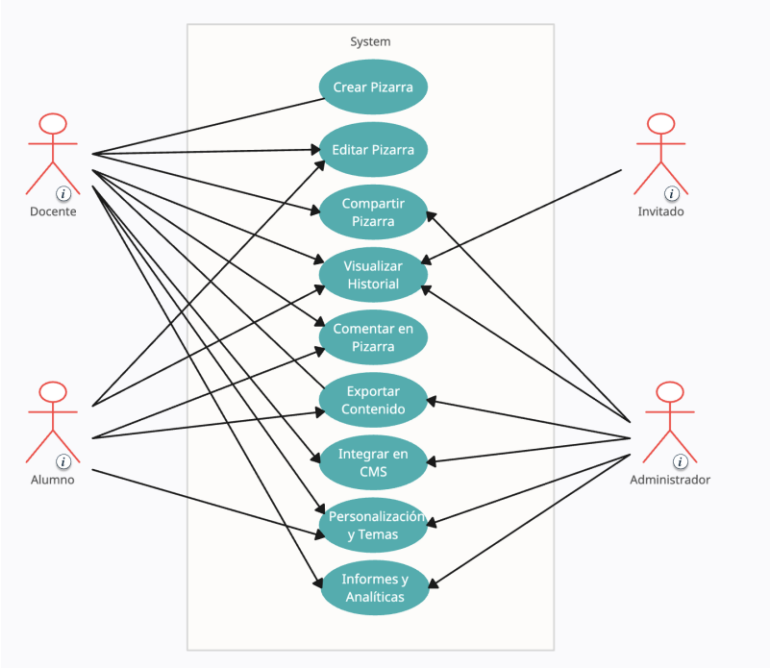
- La pizarra debe integrarse con el sistema de gestión de contenidos (CMS) existente para permitir la incrustación de pizarras en páginas web o artículos.

Personalización y Temas:

- Los usuarios deben poder personalizar la apariencia de la pizarra y seleccionar temas que se adapten a sus necesidades.

Casos de Uso

Diagrama de caso de uso



https://app.creately.com/d/start/dashboard

Diagramas de Flujo de Casos de Uso

Prioridad: requerido
Versión: 1
Complejidad: media

SP: Seleccionar Plantilla
AA: Agregar Actores
GP:

1. Crear Pizarra
Flujo: Crear pizarra
Prueba: Seleccionar nueva pizarra, elegir tipo de pizarra.

a. Crear pizarra
Flujo: SP, AA, GP

Descripción detallada de cada caso de uso

CASO No. 1 Crear Pizarra

ID:	CU-1	
Nombre	Crear Pizarra	
Actores	Docente investigador, Docente invitado	
Objetivo	Este caso debe permitir crear una pizarra	
Urgencia	5	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - La pizarra se agrega a un contenido o módulo que la admita	
Flujo Normal	Docente	Sistema
	Selecciona nueva pizarra	
		Despliega las plantillas disponibles
	Selecciona una plantilla	
	Selecciona agrega colaboradores	
		Retorna usuarios estudiantes o docentes.
	Agrega colaboradores	
	Registra la plantilla	
		Guarda la pizarra
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Exepciones		

Comentado [k1]: Son los estados finales con los que termina el caso de uso, estos se deben cumplir y cualquier situación que no permita cumplirlos debe quedar consignada como flujo alternativo.

Comentado [k2]: Son excepciones que no dependen del sistema o actores y que se deben resolver en otro flujo.

Prioridad de Requerimientos

A partir del análisis de requerimientos, funcionalidades y el proceso de design thinking, se concreta la siguiente matrix de prioridad de requerimientos.

Para la interpretación se tiene en cuenta la siguiente escala con sus valores.

Eje de Urgencia:

- Obligatoria (5)
- Alta (4)
- Moderada (3)
- Menor (2)
- Baja (1)

Eje de Esfuerzo:

- Muy alto (5)
- Alto (4)
- Medio (3)
- Bajo (2)
- Muy bajo (1)

		Urgencia				
Impacto		1-Baja	2-Menor	3-Moderada	4-Alta	5-Obligatoria
	5-Muy alto	5	10	15	20	25
	4-Alto	4	8	12	16	20
						CU-1
	3-Medio	3	6	9	12	15
	2-Bajo	2	4	6	8	10
	1-Muy bajo	1	2	3	4	5

https://asana.com/es/resources/priority-matrix

4. Requisitos No Funcionales

Seguridad:

- La pizarra debe garantizar la seguridad de los datos y la autenticación de usuarios. Debe utilizar cifrado para proteger la información.

Rendimiento:

- La aplicación debe ofrecer un rendimiento óptimo, permitiendo la colaboración en tiempo real incluso con un gran número de usuarios.

Escalabilidad:

- La pizarra debe ser escalable para manejar un aumento en el número de usuarios y la cantidad de contenido.

Disponibilidad:

- La aplicación debe estar disponible y funcionando de manera constante, minimizando el tiempo de inactividad.

Compatibilidad con Dispositivos:

- La pizarra debe ser compatible con una variedad de dispositivos, incluyendo computadoras de escritorio, tabletas y teléfonos móviles.

Usabilidad:

- La interfaz de usuario de la pizarra debe ser intuitiva y fácil de usar para usuarios de diferentes niveles de habilidad.

Accesibilidad:

- La aplicación debe ser accesible para personas con discapacidades, cumpliendo con estándares de accesibilidad web.

Cumplimiento Normativo:

- La pizarra debe cumplir con regulaciones y normativas de privacidad y seguridad de datos.

Tiempo de Respuesta:

- La aplicación debe tener tiempos de respuesta rápidos para mantener una experiencia de usuario fluida.

Requisitos de Desempeño

1. **Rendimiento en Tiempo Real**: La pizarra debe proporcionar un rendimiento en tiempo real, lo que significa que los cambios realizados por los usuarios deben reflejarse

instantáneamente para todos los colaboradores, incluso cuando múltiples usuarios trabajen simultáneamente en la pizarra. Este aspecto se debe desarrollar con sockets.

2. **Tiempo de Carga Rápido**: La pizarra debe cargar de manera eficiente, y los usuarios no deben experimentar tiempos de carga excesivamente largos al acceder a una pizarra o al editar contenido. Se requiere que los componentes estén bien diseñados y acoplados. Por lo general los componentes de la arquitectura CREAVI la cual sigue el paradigma de la programación orientada a componentes.
3. **Optimización de Recursos**: El sistema debe estar optimizado para utilizar eficientemente los recursos del servidor, minimizando el uso de CPU y memoria. El renderizado de los componentes adecuados garantiza este requisito.

Requisitos de Seguridad

4. **Acceso Seguro**: Se debe implementar una autenticación segura para garantizar que solo usuarios autorizados tengan acceso a las pizarras. Esto puede incluir autenticación de dos factores, inicio de sesión único (SSO), autenticación con JWT o Auth 2.
5. **Protección de Datos**: La pizarra debe garantizar la protección de datos sensibles, como información del usuario y contenido compartido. Se debe cifrar la información en tránsito y en reposo.
6. **Auditoría y Registro de Actividades**: El sistema debe mantener registros de actividades, lo que incluye registros de cambios en la pizarra, acceso de usuarios y eventos relevantes para la seguridad.
7. **Control de versiones**: El sistema debe llevar un registro de los cambios de los datos gestionados en la pizarra así como también los datos mismos de la estructura del componente.
8. **Variables de entorno**: El sistema debe ser manejado con variables de entorno que garanticen su fácil incorporación con otros módulos y la migración entre plataformas, así como tambien almacenar los datos iniciales del servidor como lo son las bases de datos y las llaves de autenticación, entre otras.

Requisitos de Usabilidad

9. **Interfaz Intuitiva:** La interfaz de usuario de la pizarra debe ser intuitiva y fácil de usar, permitiendo a los usuarios realizar acciones como dibujar, agregar contenido y colaborar sin dificultad.
10. **Compatibilidad con Dispositivos:** La pizarra debe ser compatible con una variedad de dispositivos, incluyendo computadoras de escritorio, tabletas y dispositivos móviles, y debe adaptarse a diferentes tamaños de pantalla.
11. **Documentación y Ayuda en Línea:** Se debe proporcionar documentación clara y ayuda en línea para los usuarios, incluyendo tutoriales y recursos de soporte.

Requisitos de Escalabilidad

12. **Manejo de Cargas Elevadas:** El sistema debe ser escalable para manejar un gran número de usuarios y múltiples pizarras simultáneamente, sin degradación significativa del rendimiento.
13. **Balanceo de Carga:** Se debe implementar un mecanismo de balanceo de carga para distribuir las solicitudes de usuarios de manera equitativa entre los servidores para garantizar la escalabilidad.
14. **Arquitectura de Backend Escalable:** La arquitectura del backend debe estar diseñada para escalar horizontalmente, lo que permite agregar más recursos de hardware a medida que aumenta la demanda.

5. Modelado E/R

Diagrama de Entidad-Relación

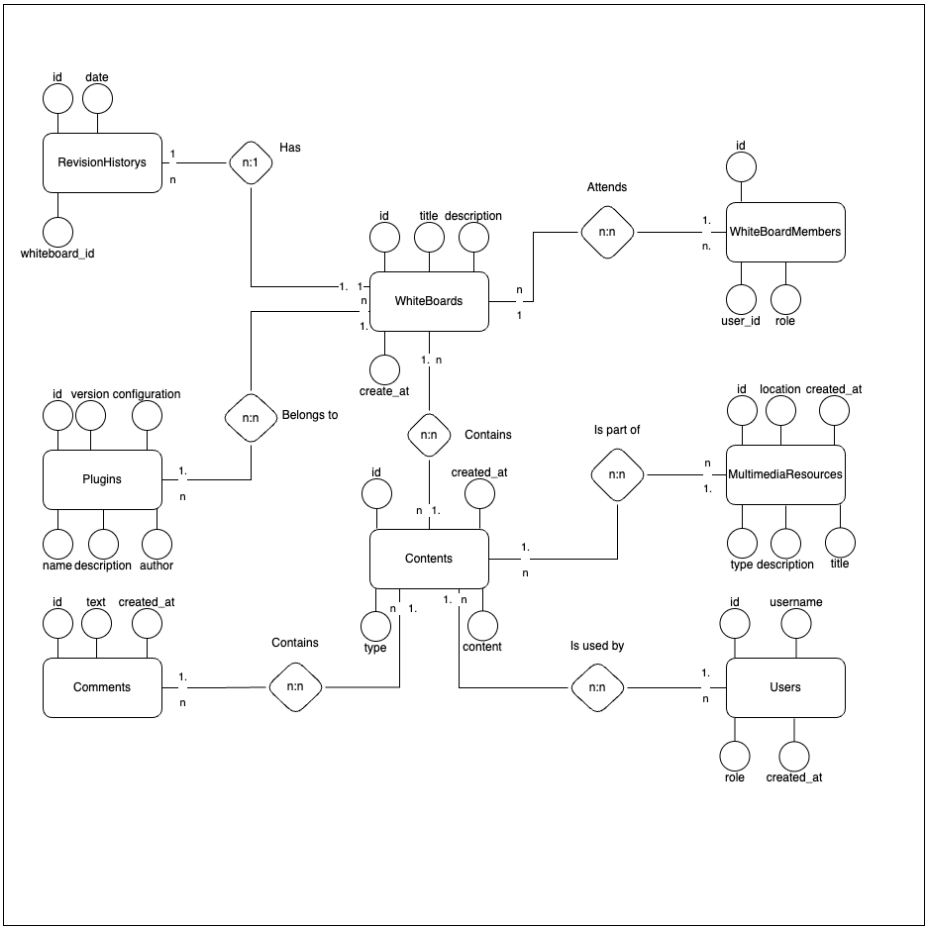
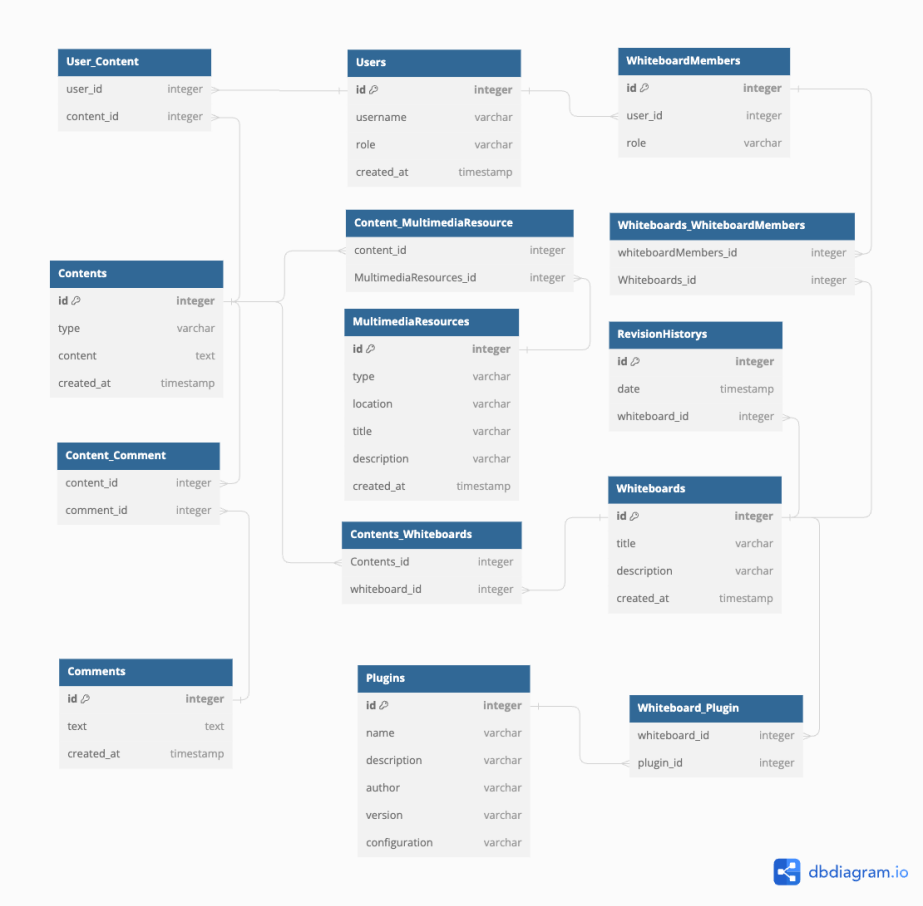


Diagrama Relacional



Script de modelo relacional

https://dbdiagram.io/

```
Table Users {
  id integer [primary key]
  username varchar
  role varchar
  created_at timestamp
}

Table User_Content {
  user_id integer [ref: > Users.id]
  content_id integer [ref: > Contents.id]
}

Table Content_Comment {
  content_id integer [ref: > Contents.id]
  comment_id integer [ref: > Comments.id]
}

Table Whiteboards {
  id integer [primary key]
  title varchar
  description varchar
  created_at timestamp
}

Table Whiteboards_WhiteboardMembers {
  whiteboardMembers_id integer [ref: > WhiteboardMembers.id]
  Whiteboards_id integer [ref: > Whiteboards.id]
}

Table Whiteboard_Plugin {
  whiteboard_id integer [ref: > Whiteboards.id]
  plugin_id integer [ref: > Plugins.id]
}

Table Contents {
  id integer [primary key]
  type varchar
  content text [note: 'Content of the content']
  created_at timestamp
}

Table Contents_Whiteboards {
  Contents_id integer [ref: > Contents.id]
  whiteboard_id integer [ref: > Whiteboards.id]
}

Table Content_MultimediaResource {
  content_id integer [ref: > Contents.id]
  MultimediaResources_id integer [ref: > MultimediaResources.id]
}

Table Comments {
  id integer [primary key]
  text text
  created_at timestamp
}

Table RevisionHistorys {
  id integer [primary key]
  date timestamp
  whiteboard_id integer [ref: > Whiteboards.id]
}

Table MultimediaResources {
  id integer [primary key]
  type varchar
  location varchar
  title varchar
  description varchar
  created_at timestamp
}

Table Plugins {
  id integer [primary key]
  name varchar
  description varchar
  author varchar
  version varchar
  configuration varchar
}

Table WhiteboardMembers {
  id integer [primary key]
  user_id integer [ref: > Users.id]
  role varchar
}
```

Descripción de Entidades y Relaciones

Entidades:

1. **User (Usuario):**
   Almacena información sobre los usuarios que pueden acceder a la pizarra.
   Atributos: ID (identificador único), nombre de usuario, rol (como administrador, docente, estudiante), fecha de creación.
   Relaciones: Cada usuario puede estar asociado con varias pizarras a través de la entidad "WhiteboardMember."
2. **Whiteboard (Pizarra):**
   Representa una pizarra en la aplicación de pizarra compartida.
   Atributos: ID (identificador único), título de la pizarra, descripción, fecha de creación.
   Relaciones: Cada pizarra puede contener contenido a través de la relación con la entidad "Content" y puede tener plugins asociados a través de la relación con la entidad "Plugin."
3. **Content (Contenido):**
   Almacena contenido que se puede agregar a las pizarras, como texto, imágenes, videos, documentos, etc.
   Atributos: ID (identificador único), tipo de contenido, contenido en sí, fecha de creación.
   Relaciones: El contenido se asocia con un usuario a través de la relación con la entidad "User" y puede estar relacionado con comentarios.
4. **Comment (Comentario):**
   Almacena comentarios realizados por los usuarios en relación con el contenido de la pizarra.
   Atributos: ID (identificador único), texto del comentario, fecha de creación.
   Relaciones: Cada comentario se relaciona con el contenido específico en la entidad "Content."
5. **RevisionHistory (Historial de Revisiones):**
   Registra el historial de revisiones y cambios realizados en las pizarras.
   Atributos: ID (identificador único), fecha de la revisión.
   Relaciones: Cada entrada de historial se relaciona con una pizarra específica en la entidad "Whiteboard."
6. **MultimediaResource (Recurso Multimedia):**
   Almacena recursos multimedia, como imágenes, videos, documentos, etc.
   Atributos: ID (identificador único), tipo de recurso, ubicación o URL del recurso, título, descripción, fecha de carga.

Relaciones: Cada entrada de Recurso multimedia se relaciona con una un contenido específico en la entidad " Content."

7. **Plugin:**
   Almacena información sobre los plugins que pueden proporcionar funcionalidades personalizadas en las pizarras.
   Atributos: ID (identificador único), nombre del plugin, descripción, autor, versión, configuración.
   Relaciones: Cada pizarra puede tener asociado uno o varios plugins a través de la entidad "Whiteboard."

Relaciones:

- "User" se relaciona con "WhiteboardMember" para indicar la asociación de los usuarios con las pizarras.
- "Content" se relaciona con "User" para registrar los usuarios que pueden interactuar con el contenido.
- "Content" se relaciona con "Comment" para permitir comentarios en los contenidos.
- "Whiteboard" se relaciona con "Content" para indicar que una pizarra puede contener contenido.
- "Whiteboard" se relaciona con "Plugin" para permitir la asociación de plugins con las pizarras.
- "WhiteboardMember" se relaciona con "User" y "Whiteboard" para indicar la asociación de usuarios con pizarras y sus roles.
- "RevisionHistory" se relaciona con "Whiteboard" para registrar revisiones en las pizarras.
- " MultimediaResource " se relaciona con "Content" para registrar recursos asociados en el contenido.

Reglas de Integridad Referencial

1. **Integridad Referencial entre "User_Content" y "Users"**: Cada registro en la tabla "User_Content" debe estar asociado con un usuario existente en la tabla "Users" a través de la clave foránea "user_id."
2. **Integridad Referencial entre "User_Content" y "Contents"**: Cada registro en la tabla "User_Content" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."

3. **Integridad Referencial entre "Content_Comment" y "Contents"**: Cada registro en la tabla "Content_Comment" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."
4. **Integridad Referencial entre "Content_Comment" y "Comments"**: Cada registro en la tabla "Content_Comment" debe estar asociado con un comentario existente en la tabla "Comments" a través de la clave foránea "comment_id."
5. **Integridad Referencial entre "Whiteboards_WhiteboardMembers" y "Whiteboards"**: Cada registro en la tabla "Whiteboards_WhiteboardMembers" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "Whiteboards_id."
6. **Integridad Referencial entre "Whiteboards_WhiteboardMembers" y "WhiteboardMembers"**: Cada registro en la tabla "Whiteboards_WhiteboardMembers" debe estar asociado con un miembro de la pizarra existente en la tabla "WhiteboardMembers" a través de la clave foránea "whiteboardMembers_id."
7. **Integridad Referencial entre "Whiteboard_Plugin" y "Whiteboards"**: Cada registro en la tabla "Whiteboard_Plugin" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "whiteboard_id."
8. **Integridad Referencial entre "Whiteboard_Plugin" y "Plugins"**: Cada registro en la tabla "Whiteboard_Plugin" debe estar asociado con un plugin existente en la tabla "Plugins" a través de la clave foránea "plugin_id."
9. **Integridad Referencial entre "Contents_Whiteboards" y "Contents"**: Cada registro en la tabla "Contents_Whiteboards" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "Contents_id."
10. **Integridad Referencial entre "Contents_Whiteboards" y "Whiteboards"**: Cada registro en la tabla "Contents_Whiteboards" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "whiteboard_id."
11. **Integridad Referencial entre "Content_MultimediaResource" y "Contents"**: Cada registro en la tabla "Content_MultimediaResource" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."
12. **Integridad Referencial entre "Content_MultimediaResource" y "MultimediaResources"**: Cada registro en la tabla "Content_MultimediaResource" debe estar asociado con un recurso multimedia existente en la tabla "MultimediaResources" a través de la clave foránea "MultimediaResources_id."
13. **Integridad Referencial entre "RevisionHistorys" y "Whiteboards"**: Cada registro en la tabla "RevisionHistorys" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "whiteboard_id."

14. **Integridad Referencial entre "WhiteboardMembers" y "Users"**: Cada registro en la tabla "WhiteboardMembers" debe estar asociado con un usuario existente en la tabla "Users" a través de la clave foránea "user_id."

Colecciones (NoSLQ)

Users: {
  _id: ObjectId,
  username: String,
  role: String,
  created_at: date,
  contents_id: [ObjectId],
}

Whiteboards: {
  _id: ObjectId,
  title: String,
  description: String,
  created_at: Date,
  contents_id: [ObjectId],
  whiteboardMembers_id: [ObjectId],
  plugin_id: [ObjectId],
}

Contents: {
  _id: ObjectId,
  type: String,
  content: String,
  created_at: Date,
  user_id: [ObjectId],
  comment_id: [ObjectId],
  whiteboard_id: [ObjectId],
  multimediaResources_id: [ObjectId],
}

WhiteboardMembers: {
  _id: ObjectId,
  user_id: ObjectId,
  role: String,
  whiteboards_id: [ObjectId],
}

Comments: {
  _id: ObjectId,
  text: String,
  created_at: ObjectId,
  contents_id: [ObjectId],
}

RevisionHistorys: {
  _id: ObjectId,
  Date: Date,
  whiteboard_id: [ObjectId],
}

MultimediaResources: {
  _id: ObjectId,
  Type: String,
  Location: String,
  Title: String,
  Description: String,
  created_at: Date,
  content_id: [ObjectId],
}

Plugins: {
  _id: ObjectId,
  Name: String,
  Description: String,
  Author: ObjectId,
  Version: String,
  Configuration: Object,
  whiteboard_id: [ObjectId],
}

6. Anexos

Diagramas Adicionales

Referencias

Etapa 2: Persistencia de Datos con Backend

7. Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

8. Diseño de la Arquitectura de Backend

Descripción de la Arquitectura Propuesta

Componentes del Backend

Diagramas de Arquitectura

9. Elección de la Base de Datos

Evaluación de Opciones (SQL o NoSQL)

Justificación de la Elección

Diseño de Esquema de Base de Datos

10. Implementación del Backend

Elección del Lenguaje de Programación

Creación de la Lógica de Negocio

Desarrollo de Endpoints y APIs

Autenticación y Autorización

11. Conexión a la Base de Datos

Configuración de la Conexión

Desarrollo de Operaciones CRUD

Manejo de Transacciones

12. Pruebas del Backend

Diseño de Casos de Prueba

Ejecución de Pruebas Unitarias y de Integración

Manejo de Errores y Excepciones

Etapa 3: Consumo de Datos y Desarrollo Frontend

13. Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

14. Creación de la Interfaz de Usuario (UI)

Diseño de la Interfaz de Usuario (UI) con HTML y CSS

Consideraciones de Usabilidad

Maquetación Responsiva

15. Programación Frontend con JavaScript (JS)

Desarrollo de la Lógica del Frontend

Manejo de Eventos y Comportamientos Dinámicos

Uso de Bibliotecas y Frameworks (si aplicable)

16. Consumo de Datos desde el Backend

Configuración de Conexiones al Backend

Obtención y Presentación de Datos

Actualización en Tiempo Real (si aplicable)

17. Interacción Usuario-Interfaz

Manejo de Formularios y Validación de Datos

Implementación de Funcionalidades Interactivas

Mejoras en la Experiencia del Usuario

18. Pruebas y Depuración del Frontend

Diseño de Casos de Prueba de Frontend

Pruebas de Usabilidad

Depuración de Errores y Optimización del Código

19. Implementación de la Lógica de Negocio en el Frontend

Migración de la Lógica de Negocio desde el Backend (si necesario)

Validación de Datos y Reglas de Negocio en el Frontend

20. Integración con el Backend

Verificación de la Comunicación Efectiva con el Backend

Pruebas de Integración Frontend-Backend

ANEXOS

Diagramas UML

- **Diagrama de Casos de Uso (Use Case Diagram):** Este diagrama muestra las interacciones entre los actores (usuarios) y el sistema. Puede ayudar a identificar las funcionalidades clave y los actores involucrados.
- **Diagrama de Secuencia (Sequence Diagram):** Estos diagramas muestran la interacción entre objetos y actores a lo largo del tiempo. Puedes utilizarlos para representar cómo los usuarios interactúan con la pizarra en un flujo de trabajo específico.
- **Diagrama de Clases (Class Diagram):** Puedes utilizar este diagrama para modelar las clases y estructuras de datos subyacentes en el sistema, como usuarios, pizarras, comentarios, revisiones, etc.

- **Diagrama de Estados (State Diagram):** Este diagrama puede ser útil para modelar el comportamiento de la pizarra en diferentes estados, como "edición", "visualización", "comentario", etc.
- **Diagrama de Despliegue (Deployment Diagram):** Puedes utilizar este diagrama para representar cómo se despliega la aplicación en servidores y cómo interactúa con otros componentes del sistema, como el CMS.
- **Diagrama de Componentes (Component Diagram):** Este diagrama puede ayudar a representar la estructura de componentes del software, como la interfaz de usuario, la lógica de negocio, las bibliotecas y los servicios utilizados.
- **Diagrama de Actividad (Activity Diagram):** Puedes usar este diagrama para modelar flujos de trabajo o procesos específicos, como el flujo de trabajo de creación y edición de contenido en la pizarra.
- **Diagrama de Comunicación (Communication Diagram):** Similar a los diagramas de secuencia, estos diagramas muestran interacciones entre objetos y actores, pero pueden ser más simples y enfocados en la comunicación.
- **Diagrama de Paquetes (Package Diagram):** Este diagrama puede ayudar a organizar y visualizar los paquetes y módulos del software, lo que es útil para el diseño modular.
- **Diagrama de Objetos (Object Diagram):** Puedes utilizar este diagrama para representar instancias de clases y cómo interactúan en un escenario específico.